

Supplementary Material for “An Efficient Privacy-Preserving Algorithm for String Search and its Application to Genomics”

Kana Shimizu^{1,4,*}

Koji Nuida^{2,3}

Gunnar Rätsch^{4,*}

¹ Biotechnology Research Institute for Drug Discovery
National Institute of Advanced Industrial Science and Technology,
2-4-7 Aomi Koto-ku, Tokyo 135-0064, Japan,

² Information Technology Research Institute,
National Institute of Advanced Industrial Science and Technology,
2-4-7 Aomi Koto-ku, Tokyo 135-0064, Japan,

³ Japan Science and Technology Agency (JST) PRESTO Researcher,
Tokyo, Japan

⁴ Computational Biology, Memorial Sloan Kettering Cancer Center,
1275 York, New York, NY, USA

S1 The sublinear communication size recursive oblivious transfer

In this section, we describe the detailed algorithm of the sublinear communication size recursive oblivious transfer. In Section 3.2, we introduced the bit-rotation technique for the case of the linear communication size oblivious transfer. As mentioned in Section 3.5, the same technique is also applied for the $O(\sqrt{N})$ -communication size oblivious transfer (SC-OT).

S1.1 The sublinear communication size oblivious transfer

Let us review the SC-OT algorithm. In the SC-OT, the one encodes the position t by in a two dimensional representation: $t_0 = t \lceil \sqrt{N} \rceil$, $t_1 = (t)_{\text{mod} \lceil \sqrt{N} \rceil}$, where $\lceil \cdot \rceil$ denotes the ceil of the argument. The user sends $\text{Enc}(t_0)$ and $\vec{\text{Enc}}(\mathbf{q})$ to the server, where

$$\vec{\text{Enc}}(\mathbf{q}) = (\text{Enc}(q_0 = 0) \dots, \text{Enc}(q_{t_1} = 1), \dots, \text{Enc}(q_{\lceil \sqrt{N} \rceil - 1} = 0)).$$

The server obtains random values r_k for $k = 0, \dots, \lceil \sqrt{N} \rceil - 1$, and computes

$$c_k = \bigoplus_{i=0}^{\lceil \sqrt{N} \rceil - 1} (v[k \times \lceil \sqrt{N} \rceil + i] \otimes \text{Enc}(q_i)) \oplus (r_k \otimes \text{Enc}(t_0 - k)),$$

and sends $\mathbf{c} = (c_0, \dots, c_{\lceil \sqrt{N} \rceil - 1})$ to the user. The user knows the result by the decryption: $\text{Dec}(c_{t_0})$. Note that $\text{Enc}(t_0 - k) = \text{Enc}(0)$ iff. $t_0 = k$, therefore the decryption of c_i becomes a random value when $i \neq t_0$. See the function SCOT in Algorithm S1 for detailed description.

S1.2 Bit-rotation technique for the sublinear communication size oblivious transfer

In this section, we will describe a new algorithm for the sublinear communication size recursive oblivious transfer (SC-ROT) by using the bit-rotation technique which is introduced in the main text. In order to apply bit-rotation technique naturally to SC-OT, the server needs to return $v[t]$ in the same two dimensional representation. The key idea here is that the server creates \mathbf{v}_0 and \mathbf{v}_1 where $v_0[i] = v[i] \lceil \sqrt{N} \rceil$ and $v_1[i] = (v[i])_{\text{mod} \lceil \sqrt{N} \rceil}$, $i = 0, \dots, N-1$, and searches on both \mathbf{v}_0 and \mathbf{v}_1 . The user obtains next t_0 and t_1 in randomized form by the search on \mathbf{v}_0 and \mathbf{v}_1 respectively using the same $\text{Enc}(t_0)$ and $\vec{\text{Enc}}(\mathbf{q})$. For the search on $v_0[i]$, the server generates random value $r_0 \in \{0, \dots, \lceil \sqrt{N} \rceil - 1\}$ and returns an encrypted value of $\hat{t}_0 = (v_0[i] + r_0)_{\text{mod} \lceil \sqrt{N} \rceil}$. For the search on $v_1[i]$, the server generates random value $r_1 \in \{0, \dots, \lceil \sqrt{N} \rceil - 1\}$ and returns an encrypted value of $\hat{t}_1 = (v_1[i] + r_1)_{\text{mod} \lceil \sqrt{N} \rceil}$. The user decrypts the server's return and obtains \hat{t}_0 and \hat{t}_1 to generate the next query $\text{Enc}(\hat{t}_0)$ and $\vec{\text{Enc}}(\hat{\mathbf{q}}) = (\text{Enc}(\hat{q}_0 = 0) \dots, \text{Enc}(\hat{q}_{\hat{t}_1} = 1), \dots, \text{Enc}(\hat{q}_{\lceil \sqrt{N} \rceil - 1} = 0))$. Since the server

*to whom correspondence should be addressed

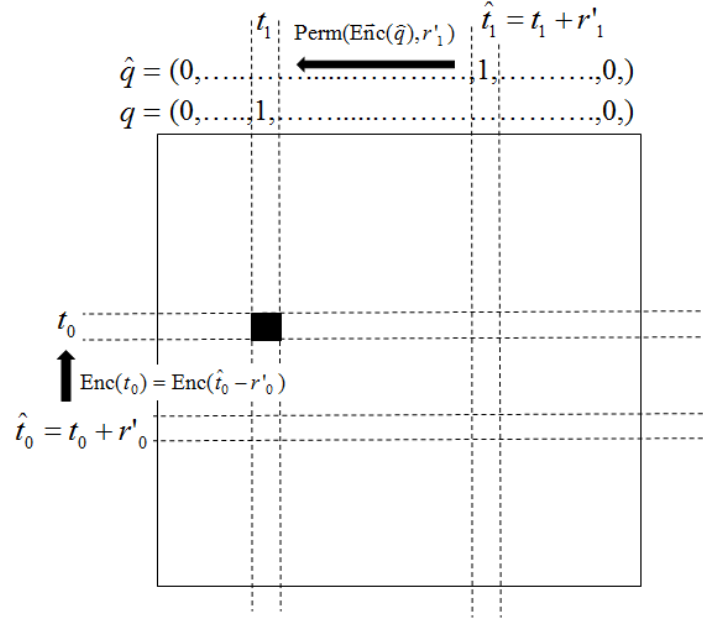


Figure S1: The illustration for the removal of random factors in the server side. \mathbf{q} and t_0 show the plain text of the user's 'true' query while $\hat{\mathbf{q}}$ and \hat{t}_0 show the plain text of the user's query. The server recovers correct t_1 by computing $-r'_1$ rotated permutation of the server's query $\hat{\mathbf{q}}$. It also recovers correct t_0 by the homomorphic encryption: $\text{Enc}(\hat{t}_0 - r'_0)$.

knows r_0 and r_1 , he/she can remove those random factors by $\text{Enc}(\hat{t}_0 - r'_0)$ and the circular bit permutation $\text{Perm}(\hat{\mathbf{q}}, r_1)$ before conducting the next search on \mathbf{v}_0 and \mathbf{v}_1 . To implement such property for the server side, we designed the server's function SCROT which is described in Algorithm S1. It takes nine arguments: user's query $\text{Enc}(\hat{t}_0)$, $\text{Enc}(\hat{\mathbf{q}})$, a vector to be searched \mathbf{v}_x ($x \in \{0, 1\}$), a random value r for randomizing the result, upper bound of the true value L_x ($x \in \{0, 1\}$), random values r'_0 and r'_1 which were used for randomizing 'true' values t_0 and t_1 in the previous round (i.e., $\hat{t}_0 = t_0 + r'_0$ and $\hat{t}_1 = (t_1 + r'_1)_{\text{mod } L_1}$) and row length L_0 and column length L_1 of the two dimensional representation (i.e., $L_0 = L_1 = \lceil \sqrt{N} \rceil$ for this case). Figure S1 illustrates the server process for removing random factors previously added to the server's return. Since $\text{Enc}(\hat{t}_0 - r'_0)$ causes the position shift from \hat{t}_0 to $(\hat{t}_0 - r'_0)_{\text{mod } L_0}$ in server's return \mathbf{c} , the server also needs another permutation $\text{Perm}(\mathbf{c}, -r'_0)$ before returning the result. See Algorithm S1 for detailed description. By this function SCROT, the server can add *removable* random factor to the result, and therefore it enables user to search \mathbf{v} recursively.

S1.3 Solving the problem caused by modulo operation of $v[i] + r$

In the function SCROT, the server generates random value $r \in \{0, \dots, \lceil \sqrt{N} \rceil - 1\}$ and conducts randomization by:

$$(v[i] + r)_{\text{mod } \lceil \sqrt{N} \rceil},$$

and returns $\text{Enc}((v[i] + r)_{\text{mod } \lceil \sqrt{N} \rceil})$ to the user.

Since the modulo operation yields different results for the same r according to the two conditions:

$$v[i] + r < \lceil \sqrt{N} \rceil$$

and

$$v[i] + r \geq \lceil \sqrt{N} \rceil,$$

and neither the user nor the server knows which condition is applied (note that the user's choice $v[i]$ and server's random value are their private information), the server needs to return two results assuming both conditions in the next round. For the case of computing $\text{Enc}(t_0)$, the sever needs to compute both

$$\mathbf{c}_0 \leftarrow \text{SCROT}(\text{Enc}(t_0), \vec{\text{Enc}}(\mathbf{q}), \mathbf{v}_0, r_0, \lceil \sqrt{N} \rceil, r'_0, r'_1, \lceil \sqrt{N} \rceil, \lceil \sqrt{N} \rceil)$$

and

$$\mathbf{c}'_0 \leftarrow \text{SCROT}(\text{Enc}(t_0), \vec{\text{Enc}}(\mathbf{q}), \mathbf{v}_0, r_0, \lceil \sqrt{N} \rceil, (r'_0 - \lceil \sqrt{N} \rceil), r'_1, \lceil \sqrt{N} \rceil, \lceil \sqrt{N} \rceil).$$

Since only one of \mathbf{c}_{0,t_0} and \mathbf{c}'_{0,t_0} becomes an encryption of a correct result and the other becomes an encryption of a random value, user is able to obtain the next t_0 by checking if $0 \leq \text{Dec}(\mathbf{c}_{0,t_0}) < \lceil \sqrt{N} \rceil$ or $0 \leq \text{Dec}(\mathbf{c}'_{0,t_0}) < \lceil \sqrt{N} \rceil$ (see the function: SCChooseDec in Algorithm S1). In similar way, the user also obtains t_1 . Algorithm S2 shows the full description of sublinear communication size recursive oblivious transfer algorithm taking into account of the above problem.

S2 The sublinear communication algorithm for $PBWT - sec$

In Section 3.3, the linear size communication algorithm for $PBWT - sec$ is introduced. Here we introduce the sublinear communication size algorithm by adapting SC-ROT to the search by $PBWT$. The goal is to find a set-longest match at a given position t between a query S and a set of genotype sequences X in a privacy-preserving manner. In this section, we consider that both t and S are private information and use the following model which is the same model as Model 3 in Section 3.4.

Model[n1] 1. *The user is a private haplotype sequence holder, and the server is a holder of a set of private haplotype sequences. The user has a vector of D positions $T = (t_1, \dots, t_D)$. The user learns nothing but a set-longest match at a given position $t \in \{t_1, \dots, t_D\}$ between the query and the database while the server learns nothing about the user's query string. The server knows T but cannot identify which element the user queries.*

Similar to the linear size communication algorithm for $PBWT - sec$, the server creates $\mathbf{v}^{(c)}$ which is a look-up vector for a letter c as follows:

$$\mathbf{v}^{(c)}[o_j + i] = \begin{cases} \text{CF}_c(P_{\cdot, (t_j+k)}) + o_j & (i = 0) \\ \text{CF}_c(P_{\cdot, (t_j+k)}) + \text{Rank}_c(P_{\cdot, (t_j+k)}, i) + o_j & (i \neq 0) \end{cases} \\ (1 \leq j \leq D, 0 \leq i \leq M)$$

where $o_j = (j-1)(M+1)$ is an offset and k is an index which is initialized by -1 and incremented by 1 in each iteration of recursive search. All those letter tables \mathbf{v}^c for $c \in \Sigma$ are concatenated into one single vector \mathbf{v} to minimize data transfer overhead. When updating the interval to extend matches by a letter $S[i]$, the user needs to specify the region of \mathbf{v} , which corresponds to a letter table $\mathbf{v}^{(S[i])}$. In our algorithm, we designed row length L_0 and column length L_1 for the two dimensional representation (L_0 and L_1 are not the matrix size of PBWT) such that elements of the same position in the different letter tables should be placed in the same column after concatenating all the tables (i.e., $(i)_{\text{mod } L_1} = (i + |\mathbf{v}^{(0)}|)_{\text{mod } L_1} = (i + |\mathbf{v}^{(0)}| + |\mathbf{v}^{(1)}|)_{\text{mod } L_1}, \dots, = (i + \sum_{c \in \{0, \dots, |\Sigma|-2\}} |\mathbf{v}^{(c)}|)_{\text{mod } L_1}$) in order that the user can specify the letter table by choosing an offset added to row value (i.e., t_0) of the query. For this purpose, the server configures $L_1 = \sqrt{D(M+1)|\Sigma|}$, an offset factor $L'_0 = \lceil D(M+1)/L_1 \rceil$, $L_0 = L'_0 \times |\Sigma|$, and extend each letter table $\mathbf{v}^{(c)}$ to the length of $L'_0 L_1$ before the concatenation to make \mathbf{v} (i.e., $|\mathbf{v}^{(0)}| = \dots = |\mathbf{v}^{(|\Sigma|-1)}| = L'_0 L_1$). To enable searching $\mathbf{v}^{(c)}$ by SC-ROT, the server converts all the elements in \mathbf{v} into the two dimensional representation and stores them in two vectors \mathbf{v}_0 and \mathbf{v}_1 each of them is of length $L'_0 L_1$. Figure S2 is a graphical view of the rearrangement of \mathbf{v}_0 and \mathbf{v}_1 .

Now the user is able to search $\mathbf{v}^{(c)}$ recursively in an oblivious manner by using SC-ROT. In PBWT, the match is reported as an interval $(f, g]$ and the number of matches is equivalent to $g - f$. Since the user wants to start the search from t_x -th column on PBWT, user initialized f and g by $f = o_x$, $g = o_x + M$ where $o_j = (j-1)(M+1)$ and computes two dimensional representation of them: $f_0 = f/L_1$, $f_1 = (f)_{\text{mod } L_1}$, $g_0 = g/L_1$, $g_1 = (g)_{\text{mod } L_1}$. Then the user recursively searches $\mathbf{v}^{(c)}$ for updating f and g until he/she finds the match. For the i -th round of the recursive search, meaning that the user updates the interval for finding matches ending with $S[i]$, he/she adds an offset $S[i]L'_0$ to f_0 and g_0 in order to specify $S[i]$. For each round, the server also computes an encrypted flag whose plain text is equal to 0 iff. $f = g$. Since there is also a similar problem caused by modulo operation discussed in the section S1.3, the server computes the encrypted flag for the case of $v[t] + r < L_0$ and $v[t] + r \geq L_0$. The detailed description of this part is described in the function isSCLongest in Algorithm S3 and item 3-(b) of Algorithm S4. Finally, the user learns the set-longest match at t by Dec(d). In order to hide the length of the set-longest match to the server, the user keep sending decoy queries until it reaches to ℓ -th round. Algorithm S3 and Algorithm S4 show a detailed algorithm of $PBWT - sec$.

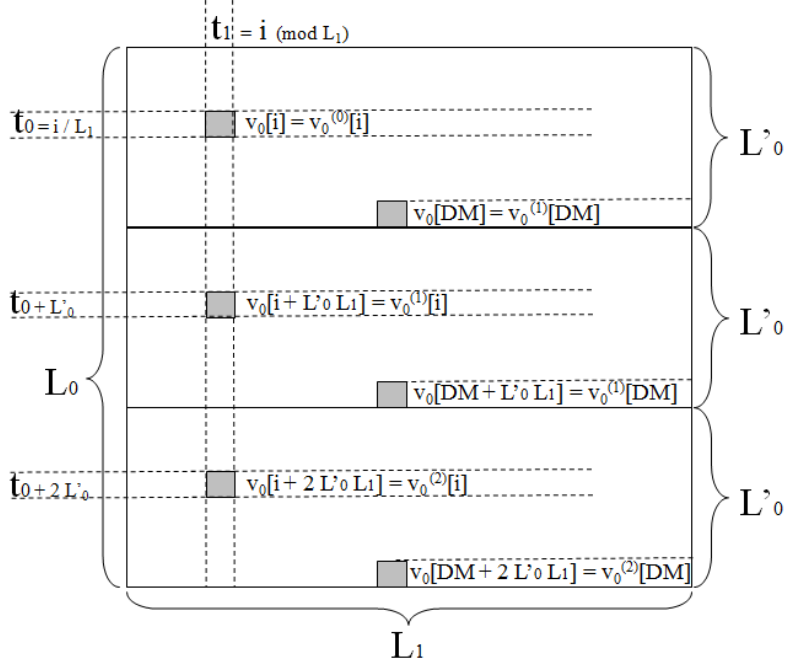


Figure S2: The arrangement of elements of \mathbf{v}_0 when $\Sigma = \{0, 1, 2\}$. The length of $\mathbf{v}_0^{(c)}$ for $c \in \Sigma$ is designed such that $v_0^{(0)}[i]$, $v_0^{(1)}[i]$ and $v_0^{(2)}[i]$ are aligned in the same column after the concatenation. The elements of \mathbf{v}_1 is also arranged in the same manner.

Algorithm S1 Building blocks for sublinear communication size recursive oblivious transfer and *PBWT* – *sec*

```

1: function SCPrepQuery( $t_0, t_1, L_1$ )
2:    $\mathbf{q} = (q_0 = 0, \dots, q_{t_1} = 1, \dots, q_{L_1-1} = 0)$ 
3:    $\vec{\text{Enc}}(\mathbf{q}) = (\text{Enc}(q_1), \dots, \text{Enc}(q_{L_1}))$ 
4:   return  $\text{Enc}(t_0), \vec{\text{Enc}}(\mathbf{q})$ 
5: end function
6:
7: function SCOT( $\text{Enc}(t_0), \vec{\text{Enc}}(\mathbf{q}), \mathbf{v}, L_0, L_1$ )
8:   for  $k = 0$  to  $L_0 - 1$  do
9:     Generate random value  $r_k$ 
10:     $x = k \times L_1$ 
11:     $c_k = \bigoplus_{i=0}^{L_1-1} (v[x+i] \otimes \text{Enc}(q_i)) \oplus r_k \otimes \text{Enc}(t_0 - k)$ 
12:   end for
13:   return  $\mathbf{c} = (c_0, \dots, c_{L_0-1})$ 
14: end function
15:
16: function SCROT( $\text{Enc}(\hat{t}_0), \vec{\text{Enc}}(\hat{\mathbf{q}}), \mathbf{v}, r, L, r'_0, r'_1, L_0, L_1$ )
17:    $\hat{\mathbf{v}} \leftarrow (\mathbf{v} + r)_{\text{mod } L}$ 
18:    $\mathbf{c} \leftarrow \text{SCOT}(\text{Enc}(\hat{t}_0 - r'_0), \text{Perm}(\vec{\text{Enc}}(\hat{\mathbf{q}}), r'_1), \hat{\mathbf{v}}, L_0, L_1)$ 
19:    $\mathbf{c} \leftarrow \text{Perm}(\mathbf{c}, -r'_0) \triangleright$  recovering the original position
20:   return  $\mathbf{c} = (c_0, \dots, c_{L_0-1})$ 
21: end function
22:
23: function SCChooseDec( $c_0, c_1, L$ )
24:   for  $x = 0$  to  $1$  do
25:      $m \leftarrow \text{Dec}(c_x)$ 
26:     if  $(0 \leq m < L)$ 
27:       return  $m$ 
28:     end if
29:   end for
30: end function

```

Algorithm S2 The detailed protocol of the sublinear communication size recursive oblivious transfer.

- Public input: the database size N , query length ℓ
- Private input of a user: a start position $t \in 0, \dots, N-1$
- Private input of a server: a vector \mathbf{v} of length N

0. (*Key setup of cryptosystem*) The user generates a key pair $(\mathbf{pk}, \mathbf{sk})$ by the key generation algorithm **KeyGen** for the additive-homomorphic cryptosystem and sends public key \mathbf{pk} to the server (the user and the server share public key \mathbf{pk} and only the user knows secret key \mathbf{sk}).
1. (*Server initialization*) The server computes $v_0[i] = v[i] \lceil \sqrt{N} \rceil$, $v_1[i] = (v[i])_{\text{mod} \lceil \sqrt{N} \rceil}$ for $i = 0, \dots, N-1$.
2. (*User initialization*) The user computes $t_0 = t \lceil \sqrt{N} \rceil$, $t_1 = (t)_{\text{mod} \lceil \sqrt{N} \rceil}$.
3. (*Recursive search*)
Initializes the index by $i \leftarrow 1$

while ($i \leq \ell$) **do**

- (a) (*Query entry*) The user performs the following steps:
 - Prepare query
if ($i \neq 1$)
 $t_0 \leftarrow \text{Dec}(c_0, t_0)$, $t_1 \leftarrow \text{Dec}(c_1, t_0)$
end if
 $\text{Enc}(t_0)$, $\vec{\text{Enc}}(\mathbf{q}) \leftarrow \text{SCPRepQuery}(t_0, t_1, \lceil \sqrt{N} \rceil)$
 - Sending $\text{Enc}(t_0)$, $\vec{\text{Enc}}(\mathbf{q})$ to the server.
- (b) (*Searching*) The server performs the following steps:
 - if** ($i \neq \ell$)
 Generating random values $r_0, r_1 \in \{0, \dots, \lceil \sqrt{N} \rceil - 1\}$
else
 $r_0 = 0$, $r_1 = 0$
end if
 \triangleright ROT removes r'_0, r'_1 from a query and add r_0 or r_1 to each result.
 $\mathbf{c}_0 \leftarrow \text{SCROT}(\text{Enc}(t_0), \vec{\text{Enc}}(\mathbf{q}), \mathbf{v}_0, r_0, \lceil \sqrt{N} \rceil, r'_0, r'_1, \lceil \sqrt{N} \rceil, \lceil \sqrt{N} \rceil)$
 $\mathbf{c}'_0 \leftarrow \text{SCROT}(\text{Enc}(t_0), \vec{\text{Enc}}(\mathbf{q}), \mathbf{v}_0, r_0, \lceil \sqrt{N} \rceil, (r'_0 - \lceil \sqrt{N} \rceil), r'_1, \lceil \sqrt{N} \rceil, \lceil \sqrt{N} \rceil)$
 $\mathbf{c}_1 \leftarrow \text{SCROT}(\text{Enc}(t_0), \vec{\text{Enc}}(\mathbf{q}), \mathbf{v}_1, r_1, \lceil \sqrt{N} \rceil, r'_0, r'_1, \lceil \sqrt{N} \rceil, \lceil \sqrt{N} \rceil)$
 $\mathbf{c}'_1 \leftarrow \text{SCROT}(\text{Enc}(t_0), \vec{\text{Enc}}(\mathbf{q}), \mathbf{v}_1, r_1, \lceil \sqrt{N} \rceil, (r'_0 - \lceil \sqrt{N} \rceil), r'_1, \lceil \sqrt{N} \rceil, \lceil \sqrt{N} \rceil)$
 $r'_0 \leftarrow r_0$, $r'_1 \leftarrow r_1$
 - Sending \mathbf{c}_0 , \mathbf{c}'_0 , \mathbf{c}_1 , \mathbf{c}'_1 to the user.

$i \leftarrow i + 1$

end while

4. (*Decryption of the result*) The user performs the following steps to obtain result x .
 $t_0 \leftarrow \text{SCChooseDec}(c_0, t_0, \mathbf{c}'_0, t_0, \lceil \sqrt{N} \rceil)$, $t_1 \leftarrow \text{SCChooseDec}(c_1, t_0, \mathbf{c}'_1, t_0, \lceil \sqrt{N} \rceil)$
 $x = t_0 \times \lceil \sqrt{N} \rceil + t_1$

Algorithm S3 Building blocks for sublinear communication size *PBWT* – *sec*

```

1: function isSCLongest(Enc( $f_0$ ), Enc( $g_0$ ),  $\vec{\text{Enc}}(\mathbf{q}_f)$ ,  $\vec{\text{Enc}}(\mathbf{q}_g)$ )
2:   for  $i = 0$  to  $L_1 - 1$  do
3:     Generating random value  $r$ 
4:      $d = d \oplus \text{Enc}(r \times (q_f[i] - q_g[i]))$ 
5:   end for
6:   Generating random value  $r$ 
7:    $d = d \oplus \text{Enc}(r \times (f_0 - g_0))$ 
8:   return  $d$ 
9: end function

10:
11: function isSCLongestGT $\epsilon$ (Enc( $f_0$ ), Enc( $g_0$ ),  $\vec{\text{Enc}}(\mathbf{q}_f)$ ,  $\vec{\text{Enc}}(\mathbf{q}_g)$ ,  $\epsilon$ )
12:   for  $k = 0$  to  $\epsilon - 1$  do ▷ For the case that  $q_g[i] = 1$  move to  $q'_g[j]$  when  $i > j$ 
13:      $\vec{\text{Enc}}(\mathbf{q}'_g) = \text{Perm}(\vec{\text{Enc}}(\mathbf{q}_g), k)$  ▷  $\mathbf{q}'_f = \text{Perm}(\mathbf{q}'_g, k)$  iff.  $(g - f) = k$ 
14:     for  $i = L_1 - k$  to  $L_1 - 1$  do
15:        $\text{Enc}(q'_g[i]) = \text{Enc}(0)$  ▷ Avoid a wrong match
16:     end for
17:     for  $i = 0$  to  $L_1 - 1$  do
18:       Generating random value  $r$ 
19:        $d = d \oplus \text{Enc}(r \times (q'_f[i] - q'_g[i]))$ 
20:     end for
21:     Generating random value  $r$ 
22:      $d_k = d \oplus \text{Enc}(r \times (f_0 - g_0))$ 
23:   end for

24:
25:   for  $k = 0$  to  $\epsilon - 1$  do ▷ For the case that  $q_g[i] = 1$  move to  $q'_g[j]$  when  $i < j$ 
26:      $\vec{\text{Enc}}(\mathbf{q}'_g) = \text{Perm}(\vec{\text{Enc}}(\mathbf{q}_g), k)$  ▷  $\mathbf{q}'_f = \text{Perm}(\mathbf{q}'_g, k)$  iff.  $(g - f) = k$ 
27:     for  $i = 0$  to  $k$  do
28:        $\text{Enc}(q'_g[i]) = \text{Enc}(0)$  ▷ Avoid a wrong match
29:     end for
30:     for  $i = 0$  to  $L_1 - 1$  do
31:       Generating random value  $r$ 
32:        $d = d \oplus \text{Enc}(r \times (q'_f[i] - q'_g[i]))$ 
33:     end for
34:     Generating random value  $r$ 
35:      $d_{\epsilon+k} = d \oplus \text{Enc}(r \times (f_0 - g_0 + 1))$ 
36:   end for
37:    $\mathbf{d} = (d_0, \dots, d_{2\epsilon})$ 
38:   Shuffling order of elements in  $\mathbf{d}$ 
39:   return  $\mathbf{d}$ 
40: end function
41:

```

Algorithm S4 The detailed description of sublinear communication size *PBWT* – *sec* for finding a set-longest match at position t .

- Public input: The length of column M , a set of alphabet letters $\Sigma = \{0, 1, \dots, |\Sigma| - 1\}$ and a set of $(D - 1)$ decoy positions and true position $T = (t_1, \dots, t_D)$.
- Private input of a user: A starting column $t_x \in T$, a query sequence S of length ℓ
- Private input of a server: PBWT matrix $P \in \mathbb{N}^{M \times N}$

0. (*Key setup of cryptosystem*) The user generates a key pair $(\mathbf{pk}, \mathbf{sk})$ by the key generation algorithm **KeyGen** for the additive-homomorphic cryptosystem and sends public key \mathbf{pk} to the server (while only the user knows secret key \mathbf{sk}).

1. (*Server initialization*)

- The server computes $L_1 = \sqrt{D(M+1)|\Sigma|}$, $L'_0 = \lceil D(M+1)/L_1 \rceil$, $L_0 = L'_0 \times |\Sigma|$ and announces L_0 , L_1 and L'_0 to the user.

2. (*User initialization*)

- The user initialize a half-open interval $(f, g]$ by $f = o_x$, $g = o_x + M$ where $o_j = (j - 1)(M + 1)$.
- The user computes two dimensional representation of $(f, g]$ by $f_0 \leftarrow f/L_1$, $f_1 \leftarrow (f)_{\text{mod } L_1}$, $g_0 \leftarrow g/L_1$, $g_1 \leftarrow (g)_{\text{mod } L_1}$

3. (*Recursive search*) Initializes the indices by $i \leftarrow 1$ $k \leftarrow -1$

while $(i \leq \ell)$ **do**

(a) (*Query entry*) The user performs the following steps:

- Prepare next query:
 $f_0 \leftarrow f_0 + S[i]L'_0$, $g_0 \leftarrow g_0 + S[i]L'_0$ \triangleright Setting offset to search matches ending with $S[i]$
 $(\text{Enc}(f_0), \vec{\text{Enc}}(\mathbf{q}_f)) \leftarrow \text{SCPRepQuery}(f_0, f_1, L_1)$, $(\text{Enc}(g_0), \vec{\text{Enc}}(\mathbf{q}_g)) \leftarrow \text{SCPRepQuery}(g_0, g_1, L_1)$
- Sending $\text{Enc}(f_0)$, $\vec{\text{Enc}}(\mathbf{q}_f)$, $\text{Enc}(g_0)$, $\vec{\text{Enc}}(\mathbf{q}_g)$, $\text{Enc}(S[i])$ to the server.

(b) (*Searching*) The server performs the following steps:

- Computes vectors $\mathbf{v}^{(c)}$ of length $D \times (M + 1)$ for all $c \in \Sigma$:

$$\mathbf{v}^{(c)}[o_j + u] = \begin{cases} \text{CF}_c(P_{\cdot, (t_j+k)}) + o_j & (u = 0) \\ \text{CF}_c(P_{\cdot, (t_j+k)}) + \text{Rank}_c(P_{\cdot, (t_j+k)}, u) + o_j & (1 \leq u \leq M) \end{cases}$$

where $o_j = (j - 1)(M + 1)$ for $j = 1, \dots, D$.

- Creates vectors $\mathbf{v}_0^{(c)}, \mathbf{v}_1^{(c)}$ of length $L'_0 \times L_1$ for $c = 0, \dots, |\Sigma| - 1$.
- Computes $\mathbf{v}_0^{(c)}[i] = \mathbf{v}^{(c)}[i]/L_1$, $\mathbf{v}_1^{(c)}[i] = (\mathbf{v}^{(c)}[i])_{\text{mod } L_1}$ for $i = 0, \dots, D(M + 1) - 1$ and $c = 0, \dots, |\Sigma| - 1$.
- Creates vectors \mathbf{v}_0 and \mathbf{v}_1 by concatenating $\mathbf{v}_0 = \mathbf{v}_0^{(0)}, \dots, \mathbf{v}_0^{(|\Sigma|-1)}$ and $\mathbf{v}_1 = \mathbf{v}_1^{(0)}, \dots, \mathbf{v}_1^{(|\Sigma|-1)}$.
- Generates random values $r_0^{(f)}, r_0^{(g)} \in \{0, \dots, L_0 - 1\}$, $r_1^{(f)}, r_1^{(g)} \in \{0, \dots, L_1 - 1\}$
- Computes next intervals and an encrypted flag showing if the match is the longest

$$\begin{aligned} \mathbf{c}_0^{(f)} &\leftarrow \text{SCROT}^\dagger(\text{Enc}(f_0), \vec{\text{Enc}}(\mathbf{q}_f), \mathbf{v}_0, r_0^{(f)}, L_0, r_0^{(f)}, r_1^{(f)}, L_0, L_1), \\ \mathbf{c}'_0^{(f)} &\leftarrow \text{SCROT}^\dagger(\text{Enc}(f_0), \vec{\text{Enc}}(\mathbf{q}_f), \mathbf{v}_0, r_0^{(f)}, L_0, (r_0^{(f)} - L_0), r_1^{(f)}, L_0, L_1), \\ \mathbf{c}_1^{(f)} &\leftarrow \text{SCROT}^\dagger(\text{Enc}(f_0), \vec{\text{Enc}}(\mathbf{q}_f), \mathbf{v}_1, r_1^{(f)}, L_1, r_0^{(f)}, r_1^{(f)}, L_0, L_1), \\ \mathbf{c}'_1^{(f)} &\leftarrow \text{SCROT}^\dagger(\text{Enc}(f_0), \vec{\text{Enc}}(\mathbf{q}_f), \mathbf{v}_1, r_1^{(f)}, L_1, (r_0^{(f)} - L_0), r_1^{(f)}, L_0, L_1), \\ \mathbf{c}_0^{(g)} &\leftarrow \text{SCROT}^\dagger(\text{Enc}(g_0), \vec{\text{Enc}}(\mathbf{q}_g), \mathbf{v}_0, r_0^{(g)}, L_0, r_0^{(g)}, r_1^{(g)}, L_0, L_1), \\ \mathbf{c}'_0^{(g)} &\leftarrow \text{SCROT}^\dagger(\text{Enc}(g_0), \vec{\text{Enc}}(\mathbf{q}_g), \mathbf{v}_0, r_0^{(g)}, L_0, (r_0^{(g)} - L_0), r_1^{(g)}, L_0, L_1), \\ \mathbf{c}_1^{(g)} &\leftarrow \text{SCROT}^\dagger(\text{Enc}(g_0), \vec{\text{Enc}}(\mathbf{q}_g), \mathbf{v}_1, r_1^{(g)}, L_1, r_0^{(g)}, r_1^{(g)}, L_0, L_1), \\ \mathbf{c}'_1^{(g)} &\leftarrow \text{SCROT}^\dagger(\text{Enc}(g_0), \vec{\text{Enc}}(\mathbf{q}_g), \mathbf{v}_1, r_1^{(g)}, L_1, (r_0^{(g)} - L_0), r_1^{(g)}, L_0, L_1), \end{aligned}$$

$$\begin{aligned} d_0 &\leftarrow \text{isSCLongest}(\text{Enc}(f_0 - r_0^{(f)}), \text{Enc}(g_0 - r_0^{(g)}), \text{Perm}(\vec{\text{Enc}}(\mathbf{q}_f), r_1^{(f)}), \text{Perm}(\vec{\text{Enc}}(\mathbf{q}_g), r_1^{(g)})) \\ d_1 &\leftarrow \text{isSCLongest}(\text{Enc}(f_0 - r_0^{(f)} + L_0), \text{Enc}(g_0 - r_0^{(g)}), \text{Perm}(\vec{\text{Enc}}(\mathbf{q}_f), r_1^{(f)}), \text{Perm}(\vec{\text{Enc}}(\mathbf{q}_g), r_1^{(g)})) \\ d_2 &\leftarrow \text{isSCLongest}(\text{Enc}(f_0 - r_0^{(f)}), \text{Enc}(g_0 - r_0^{(g)} + L_0), \text{Perm}(\vec{\text{Enc}}(\mathbf{q}_f), r_1^{(f)}), \text{Perm}(\vec{\text{Enc}}(\mathbf{q}_g), r_1^{(g)})) \end{aligned}$$

- Storing random values $r_0^{(f)} \leftarrow r_0^{(f)}$, $r_1^{(f)} \leftarrow r_1^{(f)}$, $r_0^{(g)} \leftarrow r_0^{(g)}$, $r_1^{(g)} \leftarrow r_1^{(g)}$
- Sending $\mathbf{c}_0^{(f)}, \mathbf{c}'_0^{(f)}, \mathbf{c}_1^{(f)}, \mathbf{c}'_1^{(f)}, \mathbf{c}_0^{(g)}, \mathbf{c}'_0^{(g)}, \mathbf{c}_1^{(g)}, \mathbf{c}'_1^{(g)}, d$ to the user

(c) (*Decryption of the encrypted flag and the randomized interval*) The user performs the following steps:

if $(\text{Dec}(d_0) == 0 \parallel \text{Dec}(d_1) == 0 \parallel \text{Dec}(d_2) == 0)$

Reports the result $S[1, \dots, i - 2]$ and sending the server decoy queries until $i == \ell$

else

Computes $f_0 \leftarrow \text{SCChooseDec}(\mathbf{c}_{0,f_0}^{(f)}, \mathbf{c}'_{0,f_0}^{(f)}, L'_0)$, $g_0 \leftarrow \text{SCChooseDec}(\mathbf{c}_{0,g_0}^{(g)}, \mathbf{c}'_{0,g_0}^{(g)}, L'_0)$,

$f_1 \leftarrow \text{SCChooseDec}(\mathbf{c}_{1,f_0}^{(f)}, \mathbf{c}'_{1,f_0}^{(f)}, L_1)$, $g_1 \leftarrow \text{SCChooseDec}(\mathbf{c}_{1,g_0}^{(g)}, \mathbf{c}'_{1,g_0}^{(g)}, L_1)$ \triangleright for choosing correct results

end if

$i \leftarrow i + 1$ $k \leftarrow k + 1$

end while

Algorithm S5 Building blocks for linear communication size *PBWT* – *sec*

```

1: function isLongest( $\vec{\text{Enc}}(\mathbf{q}_f)$ ,  $\vec{\text{Enc}}(\mathbf{q}_g)$ ,  $r^{(f)}$ ,  $r^{(g)}$ )
2:    $\vec{\text{Enc}}(\mathbf{q}'_f) = \text{Perm}(\vec{\text{Enc}}(\mathbf{q}_f), r^{(f)})$ 
3:    $\vec{\text{Enc}}(\mathbf{q}'_g) = \text{Perm}(\vec{\text{Enc}}(\mathbf{q}_g), r^{(g)})$ 
4:   for  $i = 0$  to  $M$  do
5:     Generating random value  $r$ 
6:      $d = d \oplus \text{Enc}(r \times (q'_f[i] - q'_g[i]))$ 
7:   end for
8:   return  $d$ 
9: end function
10:
11: function isLongestGT $\epsilon$ ( $\vec{\text{Enc}}(\mathbf{q}_f)$ ,  $\vec{\text{Enc}}(\mathbf{q}_g)$ ,  $r^{(f)}$ ,  $r^{(g)}$ ,  $\epsilon$ )
12:   for  $k = 0$  to  $\epsilon - 1$  do
13:      $\vec{\text{Enc}}(\mathbf{q}'_f) = \text{Perm}(\vec{\text{Enc}}(\mathbf{q}_f), r^{(f)})$ 
14:      $\vec{\text{Enc}}(\mathbf{q}'_g) = \text{Perm}(\vec{\text{Enc}}(\mathbf{q}_g), r^{(g)})$ 
15:      $\vec{\text{Enc}}(\mathbf{q}'_g) = \text{Perm}(\vec{\text{Enc}}(\mathbf{q}_g), k)$ 
16:     for  $i = 0$  to  $M$  do
17:       Generating random value  $r$ 
18:        $d = d \oplus \text{Enc}(r \times (q'_f[i] - q'_g[i]))$ 
19:     end for
20:      $d_k = d$ 
21:   end for
22:    $\mathbf{d} = (d_0, \dots, d_\epsilon)$ 
23:   Shuffling order of elements in  $\mathbf{d}$ 
24:   return  $\mathbf{d}$ 
25: end function

```

$\triangleright \mathbf{q}'_f = \text{Perm}(\mathbf{q}'_g, k)$ iff. $(g - f) = k$